مراجعة للمكدس والرتل واللائحة

نحتاج في الخوارزميات إلى استخدام تقنيات برمجية وأنواع معطيات كالمكدس Stack والرتل Queue واللائحة List.

يمكن برمجة هذه التقنيات من الصفر، أو الاعتماد على قوالب مكتبات جاهزة (Standard Template Library: STL) تقدمها لغة C++ للتسهيل والمرونة.

في الأمثلة الثلاثة التالية اعتمدنا على تقديم هذه التقنيات بواسطة المكتبات الجاهزة: #include <stack> و #include <queue> و #include <list>

مثال على طريقة عمل المكدس بواسطة مكتبة جاهزة هي #include <stack>

سنستخدم التوابع الجاهزة التالية:

push() يقوم بإدخال ودفع قيمة للمكدس

pop() يقوم بإخراج وسحب قيمة من المكدس

empty() يفحص فيما إذا كان المكدس فارغاً أم لا

top() يعيد القيمة الموجودة في رأس المكدس

size() يعيد عدد عناصر المكدس

```
#include <iostream>
#include <stack>
using namespace std;
void showstack(stack <int> s)
{
	while (!s.empty())
	{
		cout << '\t' << s.top();
		s.pop();
	}
	cout << '\n';
}
int main ()
{
	stack <int> s;
	s.push(10);
	s.push(30);
```

```
        s.push(20);
        s.push(5);
        s.push(1);
        cout << "The stack is : ";
        showstack(s);
        cout << "\ns.size() : " << s.size();
        cout << "\ns.top() : " << s.top();
        cout << "\ns.pop() : ";
        s.pop();
        showstack(s);
        system ("pause");
        return 0;
}
```

الخرج

```
The stack is :     1    5    20    30    10

s.size() : 5
s.top() : 1
s.pop() :     5    20    30    10
```

مثال على طريقة عمل الرتل بواسطة مكتبة جاهزة هي #include <queue>

سنستخدم التوابع الجاهزة التالية:

push() يقوم بإدخال قيمة للرتل

pop() يقوم بإخراج وسحب قيمة من الرتل

empty() يفحص فيما إذا كان الرتل فارغاً أم لا

front() يعيد القيمة الموجودة في أول الرتل

back() يعيد القيمة الموجودة في آخر الرتل

size() يعيد عدد عناصر الرتل

```
#include <iostream>
#include <queue>
using namespace std;
void showq(queue <int> gq)
{
    while (!gq.empty())
    {
        cout << '\t' << gq.front();
        gq.pop();
    }
    cout << '\n';
}
```

```
int main()
{
    queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);

    cout << "The queue gquiz is : ";
    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);
     system("pause");
    return 0;
}
```

الخرج:

```
The queue gquiz is :     10     20     30

gquiz.size() : 3
gquiz.front() : 10
gquiz.back() : 30
gquiz.pop() :     20     30
```

اللوائح المترابطة Linked List

قبل الحديث عن اللوائح يجب الحديث عن المصفوفات:

المصفوفة التقليدية: وهي ما كنا نتعامل معه في مقرر خوارزميات1. لا يمكن تغيير حجم هذه المصفوفة أثناء زمن التنفيذ.

المصفوفة الديناميكية: تعتمد على المؤشرات ويمكن تغيير حجم المصفوفة أثناء زمن التنفيذ، إلا أنها لا تدعم استخدام توابع جاهزة وبالتالي سيكون التعامل معها غير سهل.

Vector: هو نمط معطيات يشبه المصفوفة الديناميكية السابقة وموجود في مكتبة جاهزة وبالتالي يدعم استخدام توابع جاهزة لتسهيل التعامل معه.

محاسن المصفوفات والنمط الجاهز Vector: سهولة التجوال والوصول العشوائي داخل المصفوفة بواسطة دليل المصفوفة.

مساوئ المصفوفات والنمط الجاهز Vector: عملية حذف عنصر أو إضافة عنصر تتطلب معالجة كبيرة وعمليات ازاحة.

يوضح الشكل التالي عناصر مصفوفة عددها 9 عناصر.

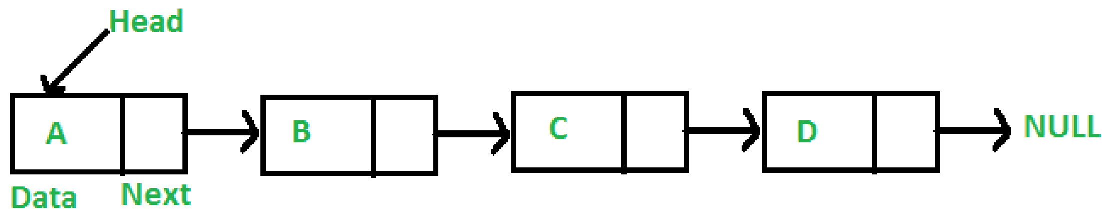
40	55	63	17	22	68	89	97	89	
0	1	2	3	4	5	6	7	8	<- Array Indices

Array Length = 9
First Index = 0
Last Index = 8

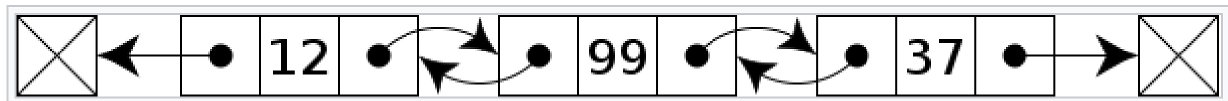
اللوائح المترابطة Linked List: هي نمط معطيات تشبه المصفوفات إلا أنه لا يتم فيها تخزين العناصر بشكل تتابعي متتالي، فهي تعتمد على مفهوم المؤشرات للربط بين العناصر.

تنقسم اللائحة المترابطة إلى أنواع أشهرها: اللائحة المفردة singly، والمزدوجة Doubled.

مثال على لائحة مترابطة مفردة: يوضح الشكل التالي أن كل عنصر (عقدة) يحوي على حقل قيمة، وعلى حقل مؤشر يشير إلى العنصر التالي.



مثال على لائحة مترابطة مزدوجة: يوضح الشكل التالي أن كل عنصر (عقدة) يحوي على حقل قيمة، وعلى حقل مؤشر يشير إلى العنصر التالي، وعلى حقل مؤشر يشير إلى العنصر السابق.



الفرق بين الدليل في المصفوفات والمؤشر iterator في اللوائح المترابطة الجاهزة:

يستخدم الدليل للتجوال على عناصر المصفوفات بينما يستخدم النمط iterator للتجوال على عناصر اللوائح في المكتبة الجاهزة.

مثال على طريقة عمل List بواسطة مكتبة جاهزة هي #include <list>

سنستخدم التوابع الجاهزة التالية:

begin(): تابع يضع مؤشر على بداية اللائحة.

end(): تابع يضع مؤشر على نهاية اللائحة.

push_front(): تابع يدخل قيمة في بداية اللائحة.

push_back(): تابع يدخل قيمة في نهاية اللائحة.

pop_front(): تابع يخرج قيمة من بداية اللائحة.

pop_back(): تابع يخرج قيمة من نهاية اللائحة.

reverse(): تابع يعكس ترتيب عناصر اللائحة.

sort(): تابع يرتب عناصر اللائحة بشكل تصاعدي.

```
#include <iostream>
#include <list>
#include <iterator>
using namespace std;

//function for printing the elements in a list
void showlist(list <int> g)
{
	list <int> :: iterator it;
	for(it = g.begin(); it != g.end(); ++it)
		cout << '\t' << *it;
	cout << '\n';
}

int main()
{
	list <int> gqlist1, gqlist2;

	for (int i = 0; i < 10; ++i)
	{
		gqlist1.push_back(i * 2);
		gqlist2.push_front(i * 3);
	}
	cout << "\nList 1 (gqlist1) is : ";
	showlist(gqlist1);

	cout << "\nList 2 (gqlist2) is : ";
	showlist(gqlist2);

	cout << "\ngqlist1.front() : " << gqlist1.front();
	cout << "\ngqlist1.back() : " << gqlist1.back();

	cout << "\ngqlist1.pop_front() : ";
```

```
	gqlist1.pop_front();
	showlist(gqlist1);

	cout << "\ngqlist2.pop_back() : ";
	gqlist2.pop_back();
	showlist(gqlist2);

	cout << "\ngqlist1.reverse() : ";
	gqlist1.reverse();
	showlist(gqlist1);

	cout << "\ngqlist2.sort(): ";
	gqlist2.sort();
	showlist(gqlist2);
	system ("pause");
	return 0;
}
```

الخرج:

```
List 1 (gqlist1) is :     0     2     4     6
8     10     12     14     16     18

List 2 (gqlist2) is :     27     24     21     18
15     12     9     6     3     0

gqlist1.front() : 0
gqlist1.back() : 18
gqlist1.pop_front() :     2     4     6     8
10     12     14     16     18

gqlist2.pop_back() :     27     24     21     18
15     12     9     6     3

gqlist1.reverse() :     18     16     14     12
10     8     6     4     2

gqlist2.sort():     3     6     9     12
15     18     21     24     27
```

المزايا الموجودة في اللوائح المترابطة والغير موجودة في المصفوفات:

- سهولة حذف العناصر بسبب المؤشرات
- سهولة اضافة العناصر بسبب المؤشرات

المساوئ الموجودة في اللوائح المترابطة والغير موجودة في المصفوفات:

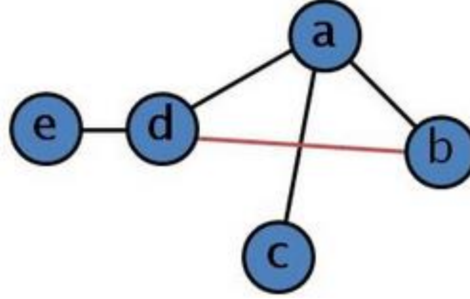
- عدم امكانية التجوال العشوائي والسريع، وانما يجب علينا الوصول تسلسلياً.
- تخصيص واستهلاك مساحات في الذاكرة لتخزين قيم المؤشرات.

تمثيل البيان برمجياً

يتم تمثيل البيان رياضياً وبالتالي برمجياً بطريقتين: مصفوفة التجاور (Adjacent Matrix) وقائمة التجاور (Adjacent List)

1. مصفوفة التجاور (Adjacency Matrix)

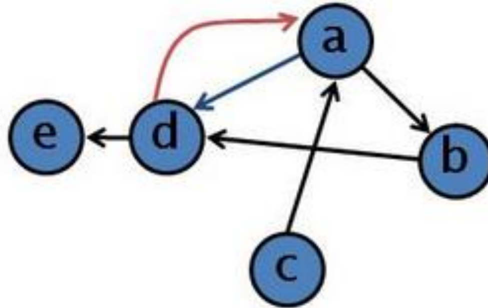
مثال: في البيان التالي أوجد مصفوفة التجاور للبيان غير الموجه.



الحل:

	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0

مثال: في الشكل التالي أوجد مصفوفة التجاور للبيان الموجه.



الحل:

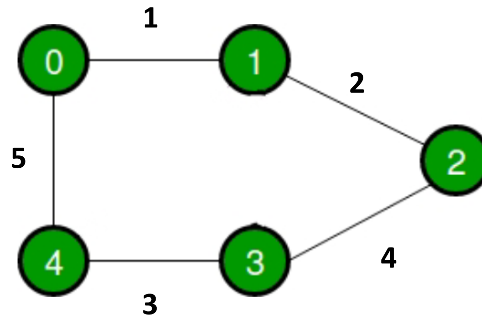
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

ملاحظة: بحال كان البيان موزون فيستبدل الرقم 1 في مصفوفة التجاور بقيمة وزن الرابط.

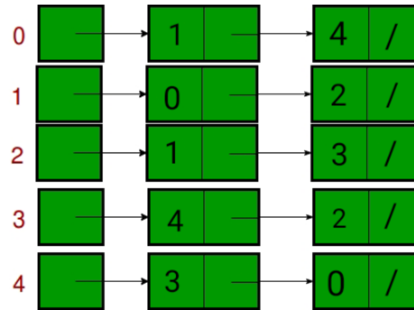
2. قائمة التجاور (Adjacency list)

ملاحظة1: __برمجياً__ يتم انشاء الروابط بين العقد بشكل متسلسل، وترتيب انشائها في البرنامج يؤثر على ترتيبها في القائمة المترابطة فالرابط المنشأ أولاً سيدخل إلى قائمة التجاور أولاً، لذا نلجأ عملياً إلى وضع أرقام على الروابط لتوضيح تسلسل انشائها.

مثال: أوجد لائحة التجاور للبيان التالي، علماً أن الأرقام الموجودة على الروابط هي لتوضيح تسلسل انشاءها:

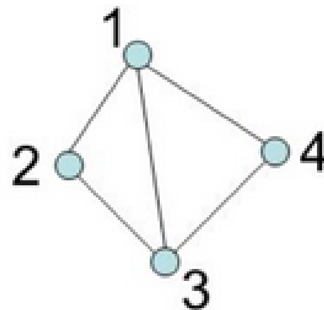


الحل:



ملاحظة2: بحال عدم كتابة أرقام على البيان تدل على تسلسل انشاء الروابط فنفترض أن العقدة ذات الرقم الأقل تكون منشأة أولاً.

مثال: أوجد قائمة التجاور للبيان التالي:



الحل:

